2D Finite-Impulse-Response (FIR) Filters

• Introduction
  - LSI digital filters are of two main types:
    - Finite-duration impulse response (FIR) or nonrecursive filters: impulse response has finite support
    - Infinite-duration impulse response (IIR) or recursive filters: input and output satisfy a difference equation of finite order
  - FIR filters:
    - Advantages:
      - Stability
      - Possibility of achieving an exact linear phase & causality
      - Efficient realization by hardware or software
    - Disadvantages:
      - Long impulse response needed to design good frequency selective filters
  - IIR filters:
    - Advantages:
      - Smaller number of parameters need to be determined when designing filter
      - Suitable for adaptive processing since one needs only to update a relatively small number of parameters.
      - Feedback
    - Disadvantages:
      - Can be unstable
      - Cannot achieve exact linear phase
2D Finite-Impulse-Response (FIR) Filters

- Impulse Response
  - An FIR filter has an impulse response $h(n_1,n_2)$ with finite support

$\begin{array}{c}
\text{Notes:} \\
\text{✓ For convenience we can assume that the region of support is rectangular; for example, one can take (smallest) rectangle enclosing the nonzero samples.} \\
\text{✓ In the multi-D case, causality is not important in general.} \\
\hspace{0.5cm} \text{– When the image samples are all available => casualty is not important} \\
\hspace{0.5cm} \text{– In case of TV signals where our data is received in time => causality would be important.}
\end{array}$
2D Finite-Impulse-Response (FIR) Filters

• Realization of 2D FIR Filters:

Let \( h(n_1, n_2) = 0 \), except for \( \begin{cases} 0 \leq n_1 \leq N_1 - 1 \\ 0 \leq n_2 \leq N_2 - 1 \end{cases} \)

\[
x(n_1, n_2) \rightarrow h(n_1, n_2) \rightarrow y(n_1, n_2)
\]

\[
y(n_1, n_2) = \sum_{l_1=0}^{N_1-1} \sum_{l_2=0}^{N_2-1} h(l_1, l_2)x(n_1 - l_1, n_2 - l_2)
\]

- 2-D convolution provides a way to implement a 2-D FIR filter (direct implementation).
- Storage: if input data come row by row (column by column), only \( N_2 \) rows (\( N_1 \) columns) are needed to be stored at a time.
2D Finite-Impulse-Response (FIR) Filters

- Filter Frequency Response

\[ H(\omega_1, \omega_2) = \text{DTFT}\{h(n_1, n_2)\} = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} h(n_1, n_2) e^{-j\omega_1 n_1} e^{-j\omega_2 n_2} \]

- The frequency response is a 2-D polynomial in \( e^{-j\omega_1} \) and \( e^{-j\omega_2} \) of degree \( (N_1 - 1) \times (N_2 - 1) \)
- These filters can be implemented directly using the convolution sum or they can be implemented using the DFT.
2D Finite-Impulse-Response (FIR) Filters

- Zero-Phase Filters
  - A filter is zero-phase if its frequency response is real
    \[ H(\omega_1, \omega_2) = H^*(\omega_1, \omega_2) \]
    \[ h(n_1, n_2) = h^*(-n_1, -n_2) \]
  - Impulse response of a zero-phase filter must have an odd-number of samples in its support with the origin at the center.
  - If \( h(n_1, n_2) \) is also real, then \( h(n_1, n_2) = h(-n_1, -n_2) \)

Each sample can be paired up with another sample of identical value. Upper side of plane maps to bottom side; i.e., only one side needed.

Number of multiplications required for implementing the filter can be reduced.
2D Finite-Impulse-Response (FIR) Filters

- Zero-Phase Filters

  - Remarks:
    - Region of support centered at origin
    - If support is $N_1 \times N_2$ then $N_1$ and $N_2$ are both odd
    - Top-Half of plane can be obtained from other half and vice-versa
    - 2-D convolution of the form
      \[
      y(n_1, n_2) = \sum_{k_1=-M_1}^{M_1} \sum_{k_2=-M_2}^{M_2} h(k_1, k_2) x(n_1 - k_1, n_2 - k_2)
      \]
      This can be reduced further if $h(n_1, n_2)$ is real-valued.

- Linear-phase filters can be easily reduced to zero-phase by discarding the Linear-phase term

- Linear-phase filters important in DSP applications (such as speech and image processing) where phase information must not be altered.
2D Finite-Impulse-Response (FIR) Filters

- Direct Implementation of FIR filters

\[ y(n_1, n_2) = \sum_{l_1=0}^{N_1-1} \sum_{l_2=0}^{N_2-1} h(l_1, l_2)x(n_1 - l_1, n_2 - l_2) \]

- An \( N_1 \times N_2 \)-point filter requires
  - \( N_1N_2 \) multiplies per output sample
  - \( N_1N_2-1 \) adds per output sample
  - \( N_2 \) rows of storage for a row-by-row implementation

- If all input samples available, we can compute the outputs in any order we desire.
- Common choices are row-by-row and column-by-column.
2D Finite-Impulse-Response (FIR) Filters

- DFT Implementation of FIR Filters
  \[ y(n_1, n_2) = x(n_1, n_2) \ast \ast h(n_1, n_2) \]
  \[ \hat{y}(n_1, n_2) = DFT^{-1}\{X(K_1, K_2).H(K_1, K_2)\} \]
  \[ \hat{y}(n_1, n_2) \] is the circular convolution of \( x(n_1, n_2) \) and \( h(n_1, n_2) \). It is a spatially aliased (replicated) version of \( y(n_1, n_2) \)
  \[ \text{If the DFT size } (L_1 \times L_2) \text{ is large enough to contain } y(n_1, n_2), \text{ then } y(n_1, n_2) = \hat{y}(n_1, n_2) \]

![Diagram](image)

\[ L_1 \geq M_1 = P_1 + N_1 - 1 \]
\[ L_2 \geq M_2 = P_2 + N_2 - 1 \]
2D Finite-Impulse-Response (FIR) Filters

• DFT Implementation of FIR Filters (continued)

\[ y(n_1, n_2) = x(n_1, n_2) * h(n_1, n_2) \]

\[ \hat{y}(n_1, n_2) = DFT^{-1}\{X(K_1, K_2).H(K_1, K_2)\} \]

➢ Computation

DFT & IDFT \[ \rightarrow \frac{2(L_1L_2 \log_2 L_1L_2)}{L_1L_2} + L_1L_2 \]

CMULTS/output sample

✓ Assumption: \( H(K_1, K_2) \) precomputed and stored

✓ Note:
  – DFT implementation independent of filter order (assuming \( P_1 \times P_2 >> N_1 \times N_2 \)).
  – Direct implementation proportional to filter order (filter size).
2D Finite-Impulse-Response (FIR) Filters

• DFT vs. Direct Implementation
  ➢ Direct (convolution) implementation more efficient when small filter size
  ➢ DFT implementation more efficient when large filter size
  ➢ DFT implementation can require significantly more storage and I/O than a direct implementation

• Block convolution methods offer compromise
2D Finite-Impulse-Response (FIR) Filters

- Block convolution implementations
  - Decompose input sequence $x(n_1, n_2)$ into blocks => limits storage
  - Convolution performed on blocks using DFT methods => maintains efficiency
  - Block convolution methods: overlap-add & overlap-save
    - Direct generalization from 1-D case (refer to Section 3.2.3 in Dudgeon & Mersereau).

<table>
<thead>
<tr>
<th>Block Size</th>
<th>Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>A block size of one corresponds to direct convolution</td>
<td></td>
</tr>
<tr>
<td>A large block size corresponds to DFT implementation</td>
<td></td>
</tr>
</tbody>
</table>

Copyright by Prof. Lina Karam